Journal of Big Data

RESEARCH Open Access

Chabok: a Map-Reduce based method to solve data warehouse problems

Mohammadhossein Barkhordari* and Mahdi Niamanesh

*Correspondence:
Barkhordari@ictrc.ac.ir
Information
and Communication
Technology Research Center,
No 5, Saeedi alley, College
intersection, Enghelab street,
Tehran 1599616313, Iran

Abstract

Currently, immense quantities of data cannot be managed by traditional database management systems. Instead, they must be managed by big data solutions using shared nothing architectures. Data warehouse systems are systems that address very large amounts of information. The most prominent data warehouse model is star schema, which consists of a fact table and some number of dimension tables. It is necessary to join the facts and dimensions for query executions on the data warehouse. In shared nothing architecture, all of the required information is not placed on a single node so it is necessary to retrieve information from other nodes, which causes network congestion and low speeds of query execution. To avoid this problem and achieve maximum parallelism, dimensions can be replicated over nodes if they are not too large. However, if there are dimensions with data volumes greater than the capacity of a node or dimensions where the data volume summation exceeds node capacity, the query execution is confronted with serious problems. In big data problems, the amount of data is immense, and thus replicating immense data cannot be considered an appropriate method. In this paper, we propose a method called Chabok, which uses two-phased Map-Reduce to solve the data warehouse problem. In this method, aggregation is performed completely on Mappers, and intermediate results are sent to the Reducer. Chabok does not need data replication for join omission. The proposed method was implemented on Hadoop, and TPC-DS queries were executed for benchmarking. The query execution time on Chabok surpassed prominent big data products for data warehousing.

Keywords: Big data, MapReduce, Data warehouse, Data locality

Introduction

Existing information is a valuable asset for many different types of organizations. Storing and analysing information can solve many problems within an organization [1]. The results from data analyses help organizations make correct decisions and provide better services for customers. Thus, high speed storage and retrieval of large volumes of data generated by electronic devices and software systems are critical issues [2–4]. Many organizations consider big data solutions because they cannot manage their data with traditional database management systems [5]; therefore, they must seek drastic measures for the design and implementation of new systems according to big data architectures. These organizations must change their architectures from

single-node to multi-node platforms. This transformation is not easy and requires a paradigm shift for data placement on different nodes [6–8].

Data warehousing is one of first and most important systems to accomplish these sophisticated changes. Because they contain the historical data of an organization, data warehouses hold large amounts of information in comparison with other systems [9, 10]. Legacy data warehouses used a single node but, due to increasing data volume and the need for high-speed query processing, shared memory and disk architectures were used. In this architecture, hardware nodes use common memory for data storage. However, this type of architecture cannot solve the query speed problem because using common memory to store data forms a bottleneck. The only remaining architecture is shared nothing architecture in which information is divided across various nodes. One of the prominent methods used in shared nothing architectures is Map-Reduce [11]. In Map-Reduce, a map function is executed on each node and then a reduce function on Reducer nodes collects intermediate Mapper results and generates the final results.

Map-Reduce is useful for data warehouse problems. Information is allocated to each Mapper and a query is then executed. In the next phase, the Reducer aggregates the results of each Mapper and creates the final results. However, using shared nothing architecture creates a new problem: the absence of data required for processing, or in other words, each node requires other nodes to execute its query. This problem is called a data locality problem, and the need to wait for other node data also causes network congestion.

The main components of data warehouse are fact, measures and dimensions. "Facts represent atomic information elements in a multi-dimensional database. A fact consists of quantifying values stored in measures and a qualifying context which is determined through (terminal) dimension levels. Each dimension level contains a set of instances or elements" [12]. "A distributive measure is a measure (i.e., function) that can be computed for a given data set by partitioning the data into smaller subsets, computing the measure for each subset, and then merging the results in order to arrive at the measure's value for the original (entire) data set" [13].

The star schema data warehouse includes a fact table and some number of dimensions. The fact table has much larger data records than the dimension tables. To fragment and allocate data warehouse information over nodes, different methods have been proposed. Some methods try to accelerate query execution by putting some metadata in each node. These methods improve query execution time but the need to exchange data among nodes remains. Other types of methods try to replicate and collocate data in order to achieve node independence. However, in big data problems, these methods make the volume of big data balloon, which is unacceptable for already immense amounts of data.

In this paper, we propose a method called Chabok that not only solves the data locality problem completely but solves network congestion problems as well. In Chabok, a two-phased Map-Reduce method is used for data warehouse problems with big data. Chabok is used for star-schema data warehouses and can compute distributive measures. This method can also be applied to big dimensions, which are dimensions where data volume is greater than the volume of a node.

Related works

In this section, we investigate related works that try to solve Map-Reduce problems related to the data warehouse. Hadoop++ [14] creates an index called Trojan. This method uses data collocation and co-partitioning to support the join operator, and join execution is done on the Mappers. Hail is another method with a shorter index length than Hadoop++. CoHadoop [15] intentionally collocates data on the nodes. Using this method, related data are placed together, and a data structure called the Locator is added to the HDFS (Hadoop file system). Using this method, Map-side join without data shuffling is possible. Llama [16] uses a columnar file (CFile) and is implemented on the HDFS. Queries in this method are only extracted from related CFiles, and it is not necessary to scan all files. Osprey [17] fragments table data between nodes, and each fragmentation is allocated to a node. Queries are divided into sub-queries and executed simultaneously on each node. GridBatch [18] is the same as CoHadoop, but colocation occurs at the file system layer. Arvand [19] is a method that integrate multi-dimensional data sources into big data analytic structure like Hadoop. NoAM [20] is an abstract model for NoSQl databases that extracts commonalities of various NoSQL systems. In [21], a method is proposed that transfers legacy data warehouses to Hive [22]. In [23], data from legacy data warehouses are transferred to Hive by a rule-based method. In [24], three physical data warehouse designs were investigated to analyse the impact of attribute distribution among column-families in HBase based on OLAP query performance. The authors conclude that OLAP query performance in HBase can be improved by using a distinct set of attribute distributions among column-families. In [25], three types of transformation are covered. In the first method, dimensions and measures are directly transferred to NOSQL (one table for each fact and dimension). In the second method, one table is transferred. Facts and dimension information are merged in that table. The last method is similar to the second method but with one difference: it uses a column family instead of a simple attribute.

In addition to the columnar format, Cheetah [26] uses compression methods. RCFile [27] uses horizontal and vertical partitioning. First, the data are partitioned horizontally, and each section is partitioned vertically. CIF [28] is a binary columnar method that first divides data horizontally, creates a directory for each partition and then creates a sub-directory for each column. A metadata file keeps directory information. MRShare [29] divides a job into queries and creates the provision that the previous execution results can be used if it is necessary to re-execute a query. ReStore [30] is a method that stores intermediate results for future calculations. In HadoopDB [31], a DBMS (Database management system) is installed on each node. Hadoop manages coordination among nodes. Using this method, it is possible to use DBMS features for local nodes. SAM [32] is a method that creates communication between Mapper nodes to decrease the query execution time. ScaDiPasi [33] uses a unified data format to create a data warehouse on information of patients. Clydesdale [34] is a method used for structured data that uses a star schema model to improve query execution. AQUA[35] is a query optimizer that manages intermediate results. AQUA uses a two-phased method to execute queries. In the first step, queries are divided into groups that can be executed together, and the results of each group are combined in the second phase. YSmart [36] is a method for converting SQL (Structured query language) into Map-Reduce jobs in which related

data are processed in a job. By using this method, the total number of jobs are decreased. In the Rope [37] method, job optimization is done by gathering statistical data about the job. These data are applied on the same job or similar jobs. BlinkDB [38] is designed for interactive query processing on immense data. The Flink [39] method is designed primarily for stream processing. It generalizes batch processing using Dataset API. Flink supports various concepts in time-based windows such as event-based processing, time-based processing, and row count-based processing. Aras [40], Atrak [41] and Hengam [42] use data unification and in-Memory database to achieve higher performance on data warehouse query execution.

Some methods use In-Memory techniques to decrease the Map-Reduce job execution time. PowerDrill [43], Shark [44], Spark and M3R [45] are prominent methods. PowerDrill is a column-based method. Shark and Spark [46] use In-memory data sets called RDD. In the case of RDD corruption, each RDD can rebuild itself by using existing data from a previous RDD. The M3R method improves Hadoop performance by omitting portions such as Heart beat or Job Tracker.

All these methods attempted to provide data locality, but none can claim to provide data locality completely. Each method tries to support data locality by changing different parts of the Map-Reduce method and improving data retrieval time. The method proposed in this paper has the following advantages in comparison with existing methods:

- Complete data locality
- Network congestion omission
- Data replication and collocation omission.

Methods

Problem definition

In this section, the proposed method that uses MapReduce to solve data warehouse problem is investigated. As it was explained in Related works section, there are many proposed methods to solve data warehouse problem for big data. But the biggest problem in all of them is data locality which is the absence of data required for processing on the same node. In the proposed method data locality problem is covered successfully and it is the main value in comparison with other prominent methods. The proposed method uses data locality to decrease query execution time, decrease network congestion and perfect use of node process power. This method can be used for data warehouses with star-schema model and distributive measures.

Chabok

In this section, we describe Chabok, our proposed method to solve the data warehouse problem with big data. The proposed method is useful for star-schema data warehouses. It can execute distributive measure functions on the data warehouse. First data from star schema must be transferred to Chabok architecture. The Chabok method uses a two-phase Map-Reduce for distributed data warehouses. Figure 1 depicts the Chabok architecture.

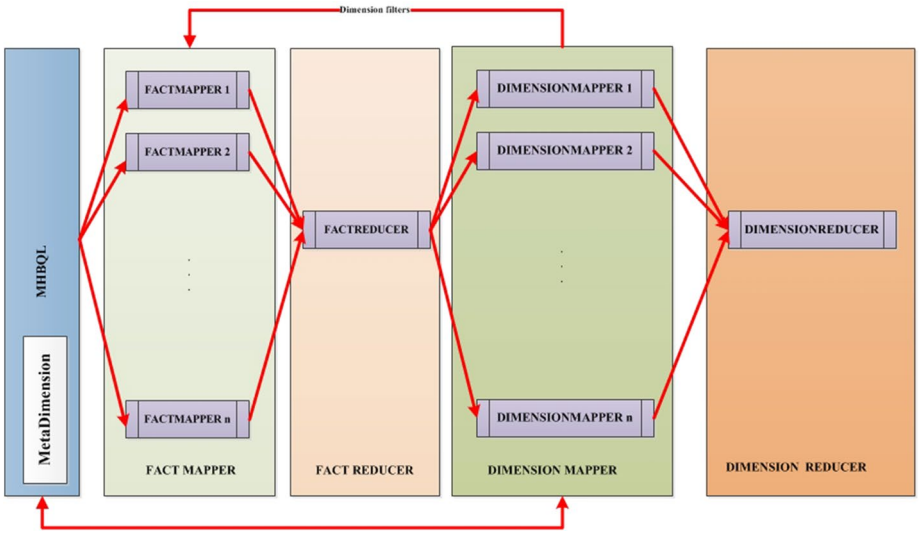


Fig. 1 Chabok architecture

The following flow-charts shows data transformation process for fact table and dimensions.

In Fig. 2 transformation, Fact fragmentation in the proposed method is a horizontal fragmentation. If there are homogeneous nodes, the same number of records is allocated to each FactMapper node. In this paper, FactMapper and DimensionMapper nodes are homogeneous.

In Fig. 3 transformation, dimension information is fragmented on one or more nodes according to its volume. Some dimensions may have small volumes, and thus it is possible to store more than one dimension on a node. This part of the architecture solves the big dimension problem that exists when large dimensions cannot be allocated to one node.

The proposed method uses two-phased Map-Reduce to solve the data warehouse problem. The first phase contains Fact table data and second phase contains dimensions data. The first MapReduce executes distributed measure functions on Mapper data. The results are aggregated on the Reducer node. If the conditions on Fact data are required, these conditions are applied on the Mapper. If the conditions on the results of distributed measure functions are required, these conditions are applied on the Reducer. To execute an input query on Chabok architecture nodes, it is necessary to have a query language. This intermediate language specifies which different conditions that are defined by users must be applied on which layers. We call this query language MHBQL. MHBQL consists of five parts:

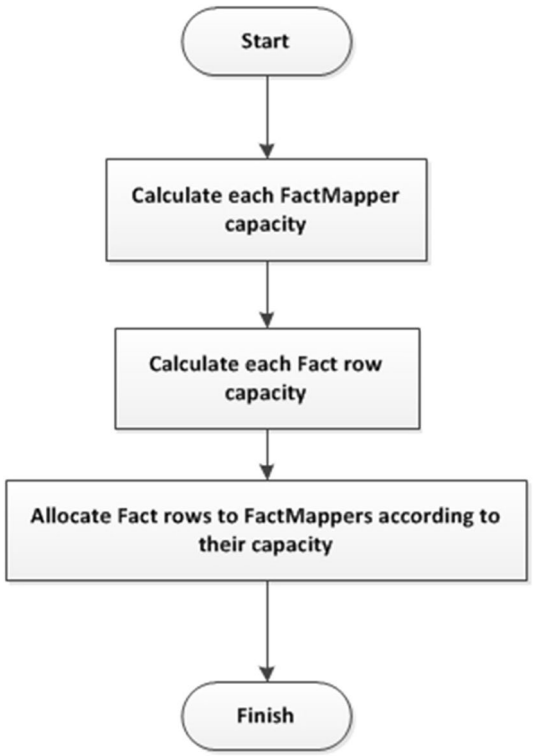


Fig. 2 Data transformation process for fact table

- Selected dimensions
  $\{Dimension_1.Attribute_1, Dimension_2.Attribute_1, \ldots, Dimension_n.Attribute_m\}$
- Distributive measures
  $\{[Distributive\ measure_1, measure_1], [Distributive\ measure_1, measure_2], \ldots, [Distributive\ measure_n, measure_m]\}$
- Conditions on dimensions
  $\{[Dimension_1.Attribute_1, operator, value], [Dimension_2.Attribute_1, operator, value], \ldots, [Dimension_n.Attribute_m, operator, value]\}$
- Conditions on measures
  $\{[measure_1, operator_1, value_1], [measure_2, operator_2, value_2], \ldots, [measure_n, operator_n, value_n]\}$
- Conditions on distributive measures
  $\{[Distributive\ measure_1\ (measure_1), operator_1, value_1], [Distributive\ measure_2\ (measure_2), operator_2, value_2], \ldots,$
  $[Distributive\ measure_n\ (measure_n), operator_n, value_n]\}$

For the *And* operator, "^" is used. For the *Or* operator, "|"is used. For priority, "/" and "\" are used.

The first four parts of MHBQL are used for FactMappers and the fifth part is used for FactReducer. Following flow-chart shows query execution process.

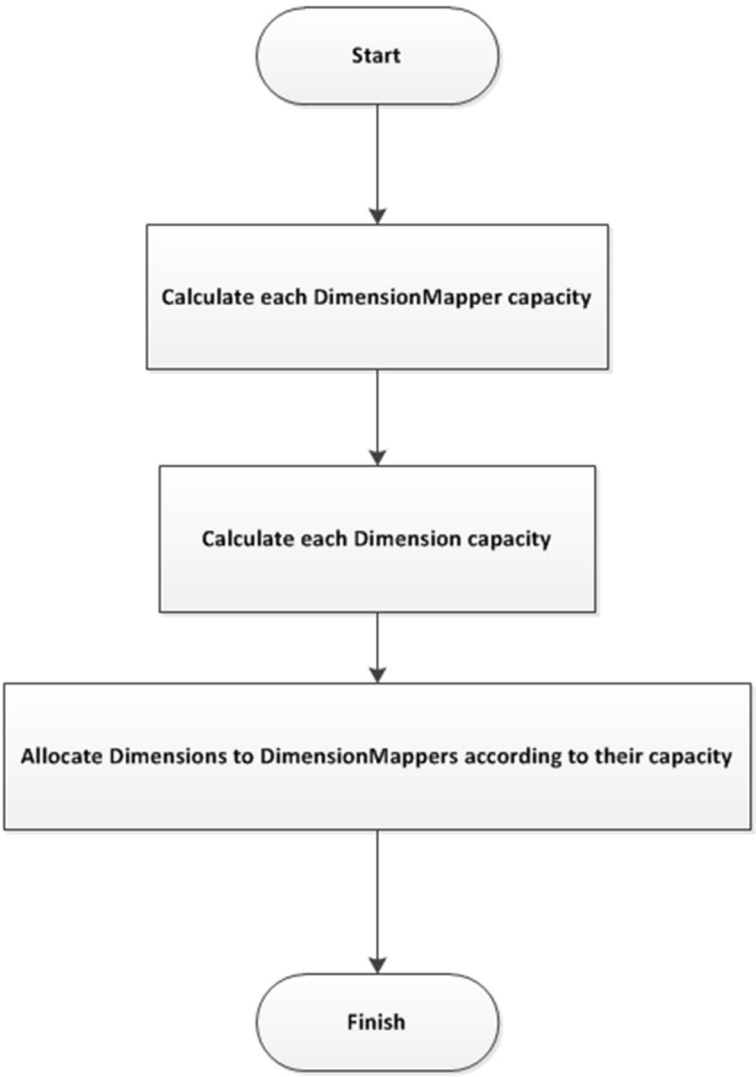


Fig. 3 Data transformation process for dimensions

Following code shows Map(FactMapper) function.

```
FactMapper
Input <Key1 , Key2 ,…, Keym , Value1 , Value2 ,… ,Valuen>
Distrubted_Measure_Function(Value1 , Value2 ,… ,Valuen)Group by (Key1 , Key2 ,…, Keym)
Filter([Value1 , Condition1], [Value2 , Condition2],…, [Valuek , Conditionk])
Output <Key1 , Key2 ,…, Keym , Value'1 , Value'2 ,… Value'n>
```

Following code shows Reduce(FactReducer) function.

```
FactReducer
Input <Key1 , Key2 ,…, Keym , Value'1 , Value'2 ,… Value'n>
Distrubted_Measure_Function(Value'1 , Value'2 ,… Value'n)Group by (Key1 , Key2 ,…, Keym)
Filter([Value'1 , Condition1], [Value'2 , Condition2],…, [Value'k , Conditionk])
Output <Key1 , Key2 ,…, Keym , Value"1 , Value"1 ,… Value"n>
```

The Second phase MapReduce is for dimensions data retrieval. In the Mapper phase each key is sent to its related Dimension node and requested data from each dimension is retrieved by join function. Following code shows Dimension Mapper function.

```
DimensionMapper
Input <Key_p, Value"_p >
Join(Key_p , Dimension_p)
Output < Dimension_p.Attribute_q, Value"_p >
```

In DimensionReducer phase the results from each DimensionMapper are placed together to generate the final results. Following code shows Dimension Reducer function.

```
DimensionReducer
Input < [Dimension_1.Attribute_1, Value"_1], [Dimension_2.Attribute_2, Value"_2],…,
[Dimension_k.Attribute_k, Value"_k]>

AddColumn([Dimension_i.Attribute_1, Value"_i],FinalResults)
Output < FinalResults >
```

Formal definitions

In this section, notations that are used in this paper are defined.Ω is used to show the dimensions set. Each dimension set has members, which are shown by ω.

$$\Omega = \{\omega_1, \omega_2, \ldots, \omega_k\}$$

Each dimension has a Key and some attributes, which are shown by ψ_{Key} and ψ_i, respectively.

$$\omega_m = \{\psi_{Key}, \psi_1, \psi_2, \ldots, \psi_n\}$$

The measure set is shown by Θ, and each member is shown by θ_i.

$$\Theta = \{\theta_1, \theta_2, \ldots, \theta_p\}$$

Distributive measures are shown by Z, and each member is shown by ζ_i.

$$Z = \{\zeta_1, \zeta_2, \ldots, \zeta_q\}$$

A Fact table is defined as including measures (θ_i), dimension keys ($\omega_i \rightarrow \psi_{Key}$) and a fact table key (ξ).

$$F = \{\theta_1, \theta_2, \ldots, \theta_r, \omega_1 \rightarrow \psi_{Key}, \omega_2 \rightarrow \psi_{Key}, \ldots, \omega_s \rightarrow \psi_{Key}, \xi\}$$

We define Λ as the operators set, α as a set with numeric and string values and β as a set with numeric values only.

$$\Lambda = \{=, >, <, \leq, \geq, \neq\}$$
$$\alpha = \{\text{string and numeric values}\}$$
$$\beta = \{\text{numeric values}\}$$

In the proposed method, there are two Map-Reduce phases: Fact Map-Reduce and Dimensions Map-Reduce. FactMappers and FactReducer are defined as μ and η, respectively.

$$M = \{\mu_1, \mu_2, \ldots, \mu_u, \eta\}$$

DimensionMappers and DimensionReducer are defined as δ and γ, respectively.

$$N = \{\delta_1, \delta_2, \ldots, \delta_v, \gamma\}$$

The fragmentation and allocation function is applied to the fact table, which allocates Fact rows to FactMappers (μ) according to the Fact key (ξ).

$$\Psi(\digamma, \xi_{Start}, \xi_{End}, \mu_w)$$

The fragmentation and allocation function is applied to the dimension table, which allocates Dimension rows to DimensionMappers (δ) according to the Dimension key (ψ).

$$\Phi\ (\omega_x, \delta_y, \psi_{StartKey}, \psi_{EndKey})$$

The user input query (Π) includes selected dimensions and distributive measures. The input query can have conditions on dimensions (Ш), measures (Γ) and distributed measures (ς).

$$\Pi(\Omega, Z(\Theta), \text{Ш}_{(\Lambda, \alpha)}(\Omega), \Gamma_{(\Lambda, \beta)}(\Theta), \varsigma_{(\Lambda, \beta)}(Z(\Theta))$$

FactMapper The map function of FactMapper uses the first four parts of MHBQL that are translated as a comprehensible query to FactMapper. The map function executes on each FactMapper. Because the input query on all FactMappers is the same, the results of the Mappers have the same format. The FactMapper function is shown as follows.

$$\partial\!\!\!/\ (Z(\Theta), \text{Ш}_{(\Lambda, \alpha)}(\Omega), \Gamma_{(\Lambda, \beta)}(\Theta), \delta_i)$$

This function has four parameters: distributive measures, conditions on measures, and conditions on fact dimension fields as well as a final parameter specifies FactMapper. Distributive measures are applied to fact measures. Conditions on measures are also applied directly to fact measures. To apply conditions to dimensions, it is necessary to first send conditions to each related dimension; the returned dimension keys are then applied to the dimension fields of the fact table as conditions.

In the Chabok architecture, the information about mapping dimension fields of fact table and dimension tables is stored in the *MetaDimension.* This table also saves information about the physical node, which stores dimension information. The *MetaDimension* is used to translate an MHBQL query into a comprehensible query for FactMapper nodes.

FactReducer In this section, the intermediate results that are produced by FactMappers are used as input for a FactReducer. The FactReducer aggregates the first phase intermediate results and produces the second phase intermediate results, which are made of dimension keys and aggregated measures.

To achieve higher computational speed, Chabok stores intermediate results in the RAM memory. We call this distributed intermediate in memory data sets *Medatum*. Each

FactMapper creates a *Medatum* and these *Medatums* are sent to FactReducer. The FactReducer aggregate *Medatums* and creates a new *Medatum*. The FactReducer (Ŧ) function is shown by the following.

$$Ŧ(Z(\Theta_\eta), \subsetneq_{(\Lambda, \beta)}(Z(\Theta_\eta)), \eta)$$

The FactReducer function has three parameters: distributed measures, conditions on result-distributed measures and the FactReducer node. Distributive measures are applied to FactMapper *Medatums*, and the final results are generated. If there are conditions on the fifth part of the input MHBQL, these conditions are applied in this section to the results of the distributive measures.

DimensionMapper As mentioned before, the generated *Medatum* from a FactReducer consists of dimension keys and the results of applying distributive measures to Fact measures. In the DimensionMapper phase, each dimension key is sent to its related DimensionMapper according to *MetaDimension* information. Then, due to the first part of MHBQL, other necessary data from each dimension is extracted. An illustrated operation is shown as follows.

$$\bowtie((\omega_i \rightarrow \psi_{Key})_\eta, \delta_1, \delta_2, \ldots, \delta_v)$$

In the proposed method, information retrieval from dimension nodes can be achieved simultaneously. This can reduce data retrieval time for big dimensions that are distributed among multiple nodes.

DimensionReducer In this part, *Medatum*s that are generated by DimensionMappers are combined with the FactReducer on dimension keys to produce the final results. The FactReducer *Medatum* contains items that are defined in the first and second parts of the input MHBQL. DimensionReducer operation is shown as follows.

$$(\Theta_\eta, \Omega_\eta, \gamma)$$

Replication To achieve replication in the proposed method, FactMappers and DimensionMappers must be replicated. In other words, if a Replica-Factor of three is required, it is necessary to copy each FactMapper and DimensionMapper node onto two other nodes. In the MetaDimension table, replication nodes for each FactMapper and DimensionMapper are determined. It is necessary to note that this replication is for data backup only and not for performance issues. The Replica-Factor can be set to one if data redundancy is not necessary.

Case study

For example, consider a bank data warehouse that has star-schema model and is built on the transactions of an EFT[1] switch. The properties of the data warehouse are as Table 1.

[1] Electronic fund transfer.

Table 1 Properties of the data warehouse

	Count	Data size (TB)
Transactions (20 years)	584,000,000,000	100
Customers	44,215,886	5
Issued cards	201,441,762	2
Accounts	145,258,359	8

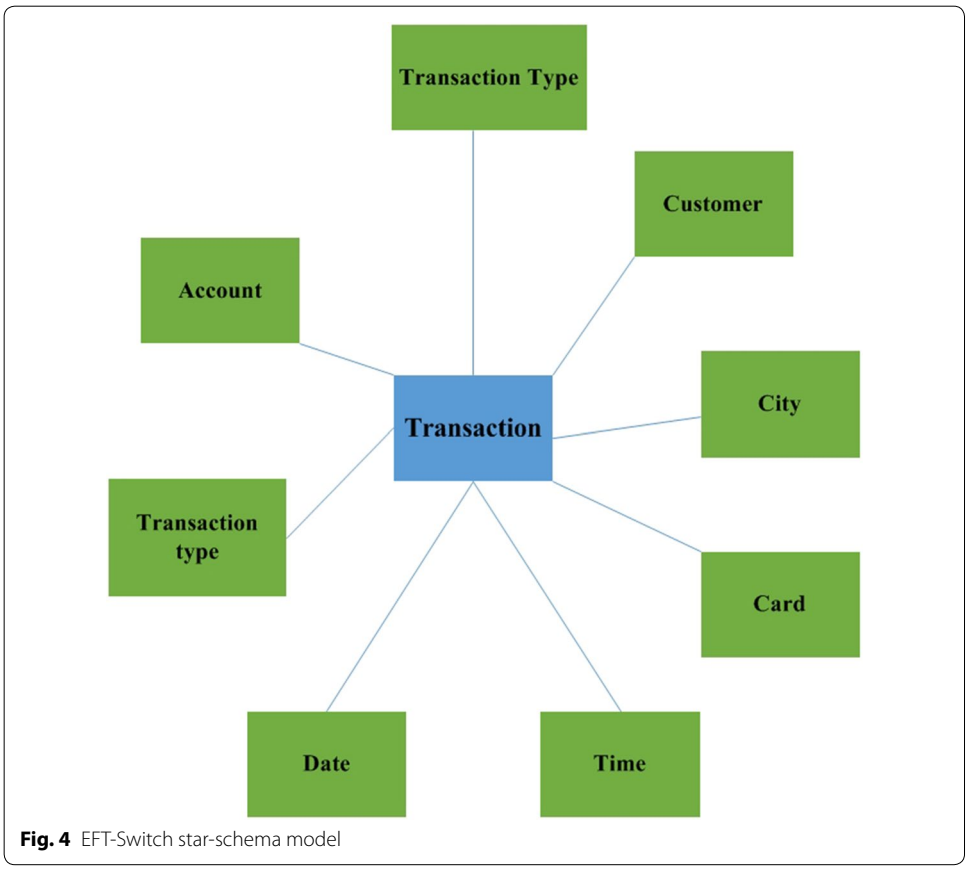


Fig. 4 EFT-Switch star-schema model

The star schema model is as Fig. 4.

According to Fig. 4 to retrieve data for most of the queries, it is required to join Transaction, Account, Customer and Card. These entities are distributed over different nodes. To join these entities it is necessary to join different parts of data on different nodes by using network connections. In prevalent methods, join process make query execution very slow. But in Chabok method, and using Fig. 5 architecture query on different nodes can be executed independently.

Tables 2, 3, 4, 5, 6, 7, 8, 9 show data structures. Input query:

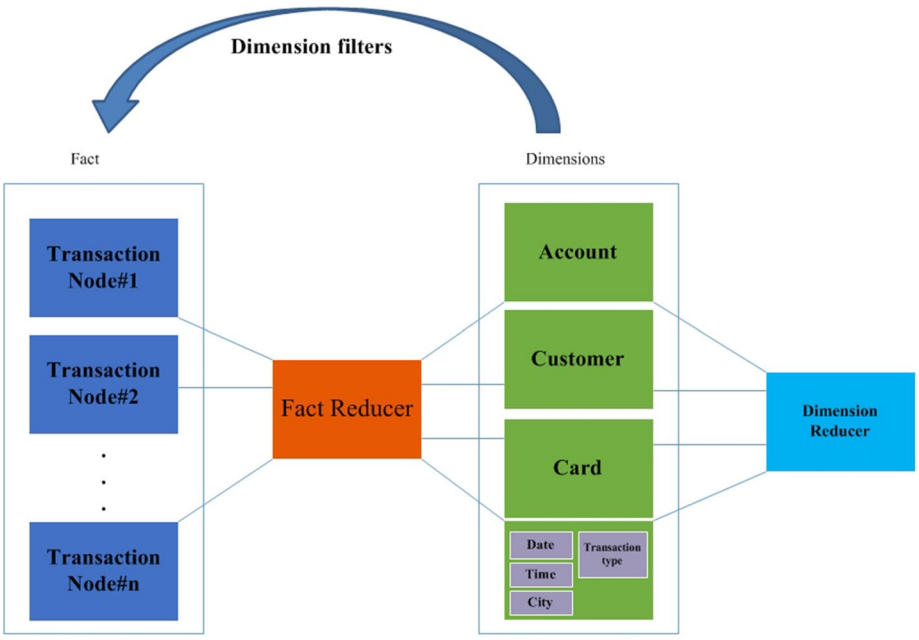


Fig. 5 Chabok case study

Table 2 Transaction

Transaction
Transaction_ID
Date
Time
Amount
Transaction_Type
Account_ID
Card_Number
Customer_Number
City ID

Table 3 City

City
City_ID
City
Province

Table 4 Transaction_Type

Transaction_Type
Transaction_Type ID
Transaction_Type_Name

Table 5 Account

Account
Account number
Open date
Account type
Branch
…

Table 6 Card

Card
Card_Number
Expire date
Card_Type
Branch
…

Table 7 Customer

Customer
Customer_Number
Name
Family
Address
Sex
Job
…

Table 8 Date

Date
Date
Day
Month
Year

Table 9 Time

Date
Time ID
Hour
Minute
Second

```
Select Sum(Tr.Amount),Count(Tr.[Transaction_ID]), Da.Month,
 tt.[Transaction_Type], Cu.Sex, Ca.[Card_Type]
From Transaction Tr  inner join Date Da on Tr. Date=Da.[Date ID]
inner join Customer Cu on Tr.[Customer_Number]=Cu.[Customer_Number]
Inner join Card Ca on Ca.[Card_Number]= Tr.[Card_Number]
Inner join Transaction_Type tt on tt.[Transaction_Type]=Tr.[Transaction_Type]
Where Tr.Amount>2000 and Count(Tr.[Transaction_ID])>350
```

The equal MHBQL is as follow.

```
{Date.Month,Customer.Sex,Card.Card_Type,Date.Month,
Transaction_Type.Transaction_Type_Name}
{[sum,Amount], [count,Transaction_ID]}
{}
{[Amount,>,2000]}
{[Count(Transaction_ID),>,350] }
```

First following query is executed on each Fact Mapper.

```
Select count(Transaction_ID), sum(Amount), Card_Number, Customer_Number, Date From
Transaction where Amount>2000
```

The results with the same format (Medatum) are aggregated on the Fact reducer. Then the conditions on the distributive measures are applied.

```
Select count(Transaction_ID), sum(Amount), Card_Number, Customer_Number, Date From
Transaction where count(Transaction_ID)>350
```

Dimension mapper is the next step. Account_ID, Card_Number, Customer_Number and Date are joined with related dimensions to extract related information.

```
Select c.Sex from customer c inner join Fact_Reducer_Medatum m on c.[Customer_Number]=
m.[Customer_Number]
Select c.[Card_Type] from Card c inner join Fact_Reducer_Medatum m on c.[Card_Number]=
m.[Card_Number]
Select d.Month from Date c inner join Fact_Reducer_Medatum m on c.[Date]= m.[Date]
```

The last step is to merge DimensionMappers Medatums to create the final results.

```
Fact_Reducer_Medatum.AddColumn(CustomerMedatum)
Fact_Reducer_Medatum.AddColumn(CardMedatum)
Fact_Reducer_Medatum.AddColumn(DateMedatum)
```

Results

In this section, the Chabok method is evaluated and compared with important open-source data warehouse products.

Experiment setup

Experimental platform

Hadoop is used for Chabok implementation and Fig. 6 shows how Hadoop is used to implement Chabok.

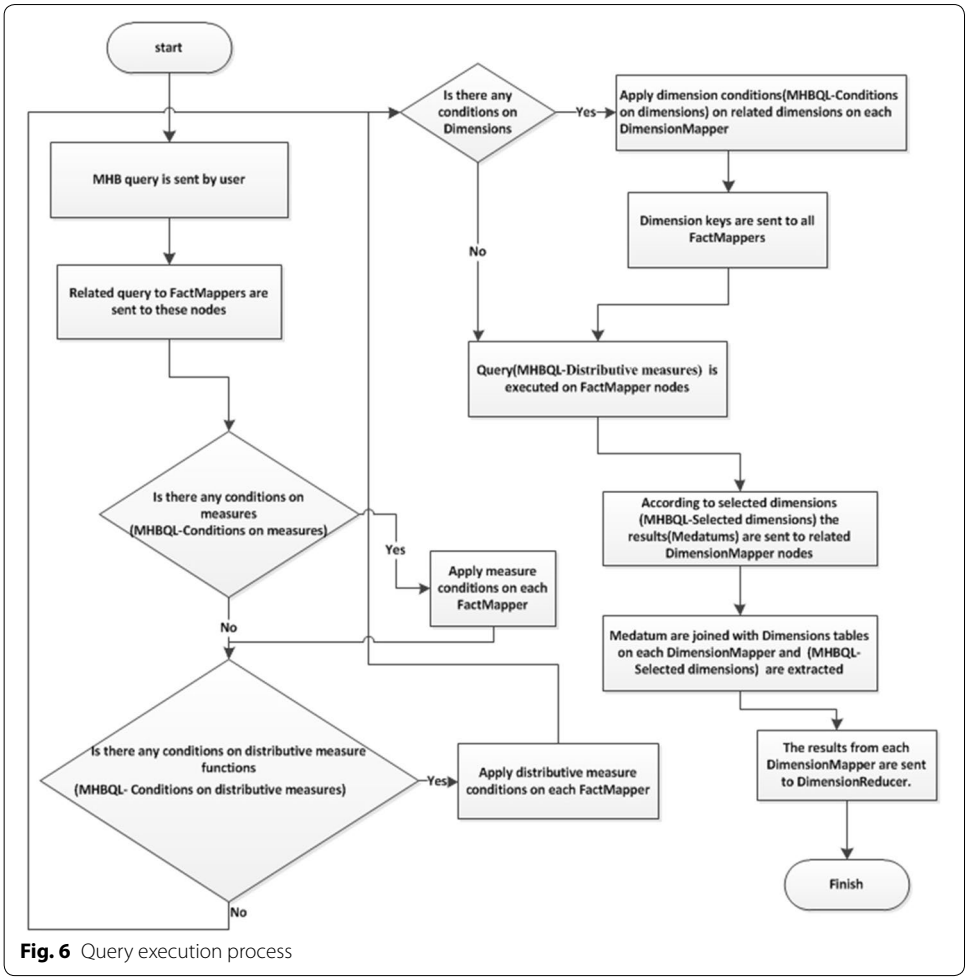


Fig. 6 Query execution process

As depicted in Fig. 7, there are three parts in the Namenode. The first part is the MHBQL parser, which translates input query to a comprehensible query for nodes. The second part is the *MetaDimension*, which stores information about the mapping fields between Fact and Dimensions. The address of the Datanode storing dimension information is also kept in the *MetaDimension*. The last part, added to the Hadoop Namenode, is the MetaNode, which stores information about the type of each Datanode. There are four types of datanodes: FactMapper, FactReducer, DimensionMapper and DimensionReducer. If the replication nodes are defined for the DimensionMappers and FactMappers, the address of replicated nodes are saved in the ReplicationNodes section.

Experiment settings

A database management system is used on each Datanode to store data. Hadoop is used as a coordinator among nodes. Ubuntu (http://www.ubuntu.com/download/server) is installed as the operating system on each node. Redis(http://redis.io/) is installed on FactReducer, DimensionReducer and Namenode nodes with *disk persistent* = *AOF* to store *Medatum*s. In addition, PostgreSQL (http://www.postgresql.org/

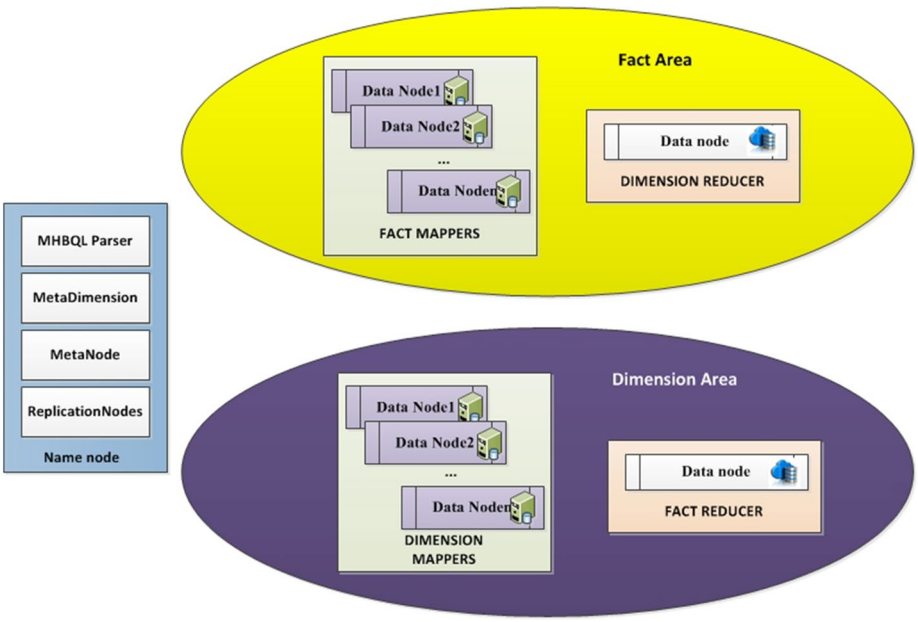


Fig. 7 Chabok implementation by Hadoop

download/linux/ubuntu/) is used to store Fact and Dimension data on FactMapper and DimensionMapper nodes.

In this section, 50 nodes were used with the specifications shown in Table 10.

To evaluate the Chabok method, we compared the data retrieval times of Hive 2.0.1 (https://hive.apache.org/downloads.html) and SparkSQL (http://spark.apache.org/) on Hadoop 2.7.3 (http://hadoop.apache.org/). Hadoop and Spark configurations are shown in Tables 11, 12, respectively.

Table 10 Node specifications

HDD	4 TB
RAM	64 GB
CPU	Dual core 3.6 GHz

Table 11 Hadoop configuration

dfs.replication	3
mapred.map.tasks	49
mapred.reduce.tasks	1

Table 12 Spark configuration

SPARK_WORKER_CORES	49

Benchmarks

For evaluation, ten TPC-DS 2.1.0 (http://www.tpc.org/tpcds/) queries were used. The proposed method is evaluated on the TPC-DS dataset. "The TPC Benchmark DS (TPC-DS) is a decision support benchmark that models several generally applicable aspects of a decision support system, including queries and data maintenance. The benchmark provides a representative evaluation of performance as a general purpose decision support system. A benchmark result measures query response time in single user mode, query throughput in multi user mode and data maintenance performance for a given hardware, operating system, and data processing system configuration under a controlled, complex, multi-user decision support workload. The purpose of TPC benchmarks is to provide relevant, objective performance data to industry users. TPC-DS Version 2 enables emerging technologies, such as Big Data systems, to execute the benchmark" (www.tpc.org/tpcds). The scale factor was equal to 100 TB and SF = 100,000. Figures 8, 9 show the ER-Diagrams.

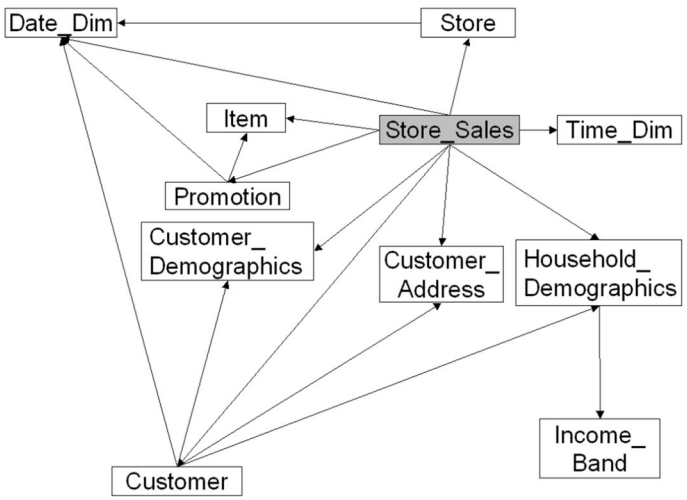


Fig. 8 Store sales diagram

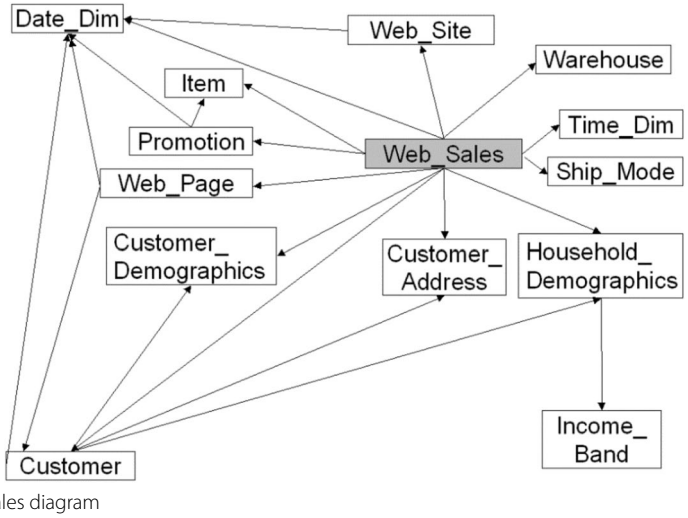


Fig. 9 Web sales diagram

Figures 10, 11, 12, 13 show the table structures.
The used queries are shown in Table 13.

Column
ss_sold_date_sk
ss_sold_time_sk
ss_item_sk (1)
ss_customer_sk
ss_cdemo_sk
ss_hdemo_sk
ss_addr_sk
ss_store_sk
ss_promo_sk

Fig. 10 Store_sales

Column
ws_sold_date_sk
ws_sold_time_sk
ws_ship_date_sk
ws_item_sk (1)
ws_bill_customer_sk
ws_bill_cdemo_sk
ws_bill_hdemo_sk
ws_bill_addr_sk
ws_ship_customer_sk
ws_ship_cdemo_sk
ws_ship_hdemo_sk
ws_ship_addr_sk
ws_web_page_sk
ws_web_site_sk
ws_ship_mode_sk
ws_warehouse_sk
ws_promo_sk

Fig. 11 web_sales

Column
i_item_sk
i_item_id (B)
i_rec_start_date
i_rec_end_date
i_item_desc
i_current_price
i_wholesale_cost
i_brand_id
i_brand
i_class_id
i_class
i_category_id
i_category
i_manufact_id
i_manufact
i_size
i_formulation
i_color
i_units
i_container
i_manager_id
i_product_name

Fig. 12 Items

Column
cd_demo_sk
cd_gender
cd_marital_status
cd_education_status
cd_purchase_estimate
cd_credit_rating
cd_dep_count
cd_dep_employed_count
cd_dep_college_count

Fig. 13 Customer_Demographics

Table 13 Node specifications

Query 8	Compute the net profit of stores located in 400 Metropolitan areas with more than 10 preferred customers Qualification substitution parameters ☐ ZIP.01 = 24128 ZIP.81 = 57834 ZIP.161 = 13354 ZIP.241 = 15734 ZIP.321 = 78668 ☐ ZIP.02 = 76232 ZIP.82 = 62878 ZIP.162 = 45375 ZIP.242 = 63435 ZIP.322 = 22245 ☐ ZIP.03 = 65084 ZIP.83 = 49130 ZIP.163 = 40558 ZIP.243 = 25733 ZIP.323 = 15798 …
Query 19	Select the top revenue generating products bought by out of zip code customers for a given year, month and manager. Qualification substitution parameters ☐ MANAGER.01 = 8 ☐ MONTH.01 = 11 ☐ YEAR.01 = 1998
Query 38	Display count of customers with purchases from all 3 channels in a given year Qualification substitution parameters ☐ DMS.01 = 1200
Query 41	How many items do we carry with specific combinations of color, units, size and category Qualification substitution parameters ☐ MANUFACT.01 = 738 ☐ SIZE.01 = medium ☐ SIZE.02 = extra large ☐ SIZE.03 = N/A ☐ SIZE.04 = small ☐ SIZE.05 = petite ☐ SIZE.06 = large ☐ UNIT.01 = Ounce ☐ UNIT.02 = Oz ☐ UNIT.03 = Bunch ☐ UNIT.04 = Ton ☐ UNIT.05 = N/A ☐ UNIT.06 = Dozen …
Query 42	For each item and a specific year and month calculate the sum of the extended sales price of store transactions Qualification substitution parameters ☐ MONTH.01 = 11 ☐ YEAR.01 = 2000
Query 45	Report the total web sales for customers in specific zip codes, cities, counties or states, or specific items for a given year and quarter Qualification substitution parameters ☐ QOY.01 = 2 ☐ YEAR.01 = 2001 ☐ GBOBC = ca_city
Query 51	Report the total of extended sales price for all items of a specific brand in a specific year and month Qualification substitution parameters ☐ MONTH.01 = 11 ☐ YEAR.01 = 2000
Query 55	For a given year, month and store manager calculate the total store sales of any combination all brands Qualification Substitution Parameters ☐ MANAGER.01 = 28 ☐ MONTH.01 = 11 ☐ YEAR.01 = 1999
Query 82	Find customers who tend to spend more money (net-paid) on-line than in stores Qualification substitution parameters MANUFACT_ID.01 = 129 MANUFACT_ID.02 = 270 MANUFACT_ID.03 = 821 MANUFACT_ID.04 = 423 INVDATE.01 = 2000-05-25 PRICE.01 = 62
Query 84	List all customers living in a specified city, with an income between 2 values Qualification substitution parameters INCOME.01 = 38128 CITY.01 = Edgewood

Table 13 (continued)

Query 98	Report on items sold in a given 30 day period, belonging to the specified category Qualification substitution parameters ☐ YEAR.01 = 1999 ☐ SDATE.01 = 1999-02-22 ☐ CATEGORY.01 = Sports ☐ CATEGORY.02 = Books ☐ CATEGORY.03 = Home

Results and analysis

This section discloses Query execution time, Query execution time, Load balancing, Network congestion and Scalability.

Query execution time

Table 14 shows the comparison results of each method. This table shows query execution times in seconds.

Figure 14 shows the query execution times in seconds.

Chabok methods have better query execution times compared to other prevailing methods. This is mainly due to complete data locality by the Mappers. In other words, each node executes its process independently and without the need of other nodes. The independent processing avoids network congestion since nodes do not need to wait to receive data. Therefore, node processing inefficiencies due to not use of maximum memory capacity and processing power do not occur. In addition, because of

Table 14 TPC-DS query execution times comparison

	Q8	Q19	Q38	Q41	Q42	Q45	Q52	Q55	Q84	Q98
Hive	750	430	440	559	638	650	680	591	603	493
SparkSQL	91	53	49	62	60	71	91	62	81	48
Chabok	21	18	17	22	21	27	32	19	29	18

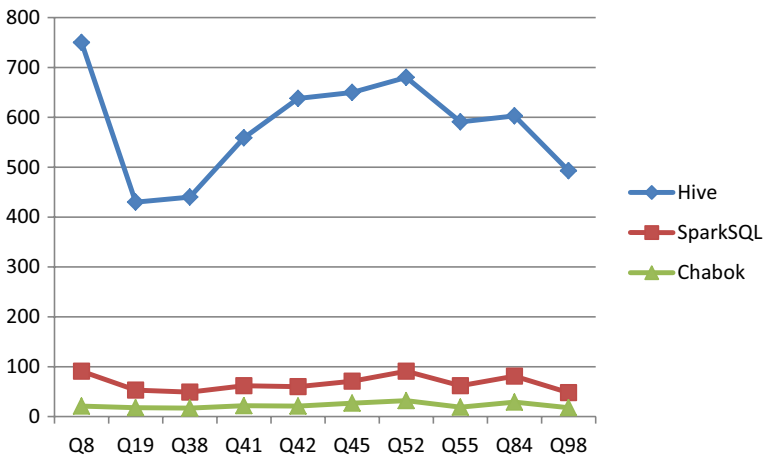


Fig. 14 TPC-DS queries execution time comparison

node independence, DBMS can be used on each node to decrease I/O and increase query execution speed when in-memory databases are used. Query execution time is decreased by using the Chabok method compared to evaluable, available and prevalent methods.

Load balancing

In this section, the load balancing of various methods is investigated. To calculate the load balance, the length of time during which a processor has a CPU usage of over eighty percent while other CPUs have CPU usages of under thirty percent is calculated. In this paper, this measure is called Balance_Factor. Using this definition, the results in Table 15 were obtained.

As Fig. 15 shows, Chabok has the lowest Load balancing factor among nodes, because for query execution it does not need other nodes data and each node can complete its query execution independently.

Network congestion

In this section the average size of data packets that were exchanged between nodes are calculated. Table 16 shows the results.

Table 15 Load balancing

Method	Balance_Factor (s)
Hive	1080
Spark-SQL	208
Chabok	22

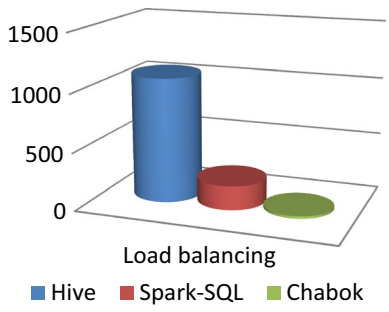


Fig. 15 Load balancing

Table 16 Network congestion

Method	Data size (MB)
Hive	2241
Spark-SQL	1855
Chabok	204

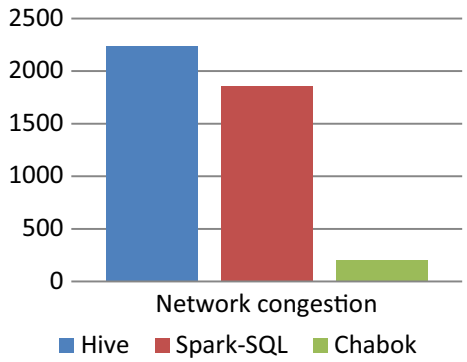
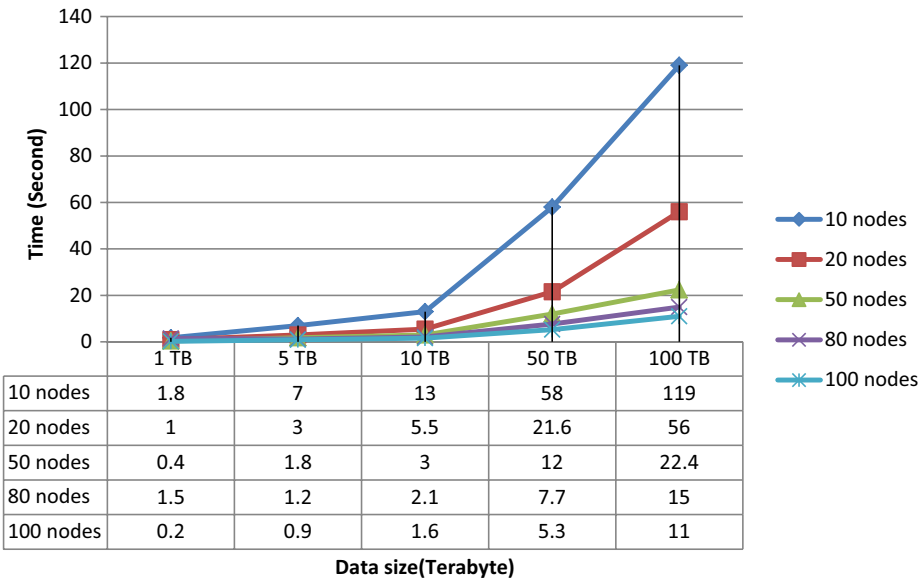


Fig. 16 Network congestion

As Fig. 16 shows, Chabok has the lowest packet exchange among nodes, because for query execution it does not need other nodes data and each node can complete its query execution independently.

Scalability

In this section Chabok scalability is evaluated. Different data sizes(TPC-DS), various number of nodes and average of query (Table 14) execution time are investigated and the results are depicted as Fig. 17. The results show that by adding more nodes and scale out data over nodes less time is required for data processing.



	1 TB	5 TB	10 TB	50 TB	100 TB
10 nodes	1.8	7	13	58	119
20 nodes	1	3	5.5	21.6	56
50 nodes	0.4	1.8	3	12	22.4
80 nodes	1.5	1.2	2.1	7.7	15
100 nodes	0.2	0.9	1.6	5.3	11

Fig. 17 Chabok scalability

Conclusion

In this paper, a method called Chabok is proposed to manage a data warehouse for big data. The proposed method can store a star schema data warehouse model and can compute distributive measures. Hardware node independency enables higher speeds of query execution, which omits joins and removes network congestion. When the proposed method was compared with Hive and Spark-SQL, it achieved a lower query execution time because of simultaneous and independent query execution on each node.

In the proposed method, there is no need for data replication or collocation for performance issues. This is very important for big data because replicating or collating big data produces an immense volume of data that make problems more sophisticated and difficult. Therefore, these kinds of solutions are not very useful. In Chabok, replication is only used for maintenance and backup issues, not for performance issues, and thus, replication is completely optional.

Finally, Chabok solves the big dimension problem, which arises when dimensions cannot fit on a single node because their volume of data exceeds that of the node. Using Chabok, dimension data can be allocated to multiple nodes without concerns about replication and collocation.

Authors' contributions
Both authors contributed equally. Both authors read and approved the final manuscript.

Acknowledgements
None.

Competing interests
The authors declare that they have no competing interests.

Consent for publication
Not applicable.

Availability of data and materials
http://www.tpc.org/TPC_Documents_Current_Versions/download_programs/tools-download-request.asp?bm_type=TPC-DS&bm_vers=2.7.0&mode=CURRENT-ONLY.

Ethics approval and consent to participate
Not applicable (No human participants).

Funding
None.

Publisher's Note

Received: 6 April 2018 Accepted: 14 September 2018
Published online: 26 October 2018

References

1. Gunasekaran A, Papadopoulos T, Dubey R, Wamba SF, Childe SJ, Hazen B, Akter S. Big data and predictive analytics for supply chain and organizational performance. J Bus Res. 2017;70:308–17.
2. Liao J, Gerofi B, Lien GY, Nishizawa S, Miyoshi T, Tomita H, Ishikawa Y. Toward a general I/O arbitration framework for netCDF based big data processing. In: European conference on parallel processing. Cham: Springer; 2016. p. 293–305.
3. Singh D, Reddy CK. A survey on platforms for big data analytics. J Big Data. 2015;2(1):8.
4. Islam NS, Shankar D, Lu X, Wasi-Ur-Rahman M, Panda DK. Accelerating I/O performance of big data analytics on HPC clusters through RDMA-based key-value store. In: 2015 44th international conference on parallel processing (ICPP). Piscataway: IEEE; 2015. p. 280–289.
5. Krishnan K. Data warehousing in the age of big data. Newnes. 2013.
6. Naresh P, Shekhar GN, Kumar MK, Rajyalakshmi P. Implementation of multi-node clusters in column oriented database using HDFS. Empirical Research Press Ltd. 2017; p. 186.

7. Azqueta-Alzúaz A, Patiño-Martinez M, Brondino I, Jimenez-Peris R. Massive data load on distributed database systems over HBase. In: 17th IEEE/ACM international symposium on cluster, cloud and grid computing (CCGRID), 2017. Piscataway: IEEE; 2017. p. 776–779.
8. Yin Z, Lan H, Tan G, Lu M, Vasilakos AV, Liu W. Computing platforms for big biological data analytics: perspectives and challenges. Comput Struct Biotechnol J. 2017;15:403–11.
9. Gad I, Manjunatha BR. Hybrid data warehouse model for climate big data analysis. In: international conference on circuit, power and computing technologies (ICCPCT), 2017. Piscataway: IEEE; 2017. p. 1–9.
10. Villegas-Ch W, Luján-Mora S, Buenaño-Fernandez D, Palacios-Pacheco X. Big data, the next step in the evolution of educational data analysis. In: international conference on information. Cham: Springer; 2018. p. 138–147.
11. Jeffrey D, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM. 2008;51(1):107–13.
12. Hüsemann B, Lechtenbörger J, Vossen G. Conceptual data warehouse design. Angewandte Mathematik und Informatik: Universität Münster; 2000. p. 1–6.
13. Jiawei H, Micheline K. Data mining, concepts and techniques, 2007.
14. Dittrich J, Quiané-Ruiz J-A, Jindal A, Kargin Y, Setty V, Schad J. Hadoop++. Proc VLDB Endow. 2010;3(1–2):515–29.
15. Eltabakh MY, Tian Y, Özcan F, Gemulla R, Krettek A, McPherson J. CoHadoop: flexible data placement and its exploitation in Hadoop. Proc VLDB Endow. 2011;4(9):575–85.
16. Lin Y, Agrawal D, Chen C, Ooi BC, Wu S. Llama: leveraging columnar storage for scalable join processing in the MapReduce framework. In: Proceedings of the 2011 ACM SIGMOD international conference on management of data. New York: ACM; 2011. p. 961–972.
17. Yang C, Yen C, Tan C, Madden SR. Osprey: implementing MapReduce-style fault tolerance in a shared-nothing distributed database. In: IEEE 26th international conference on data engineering (ICDE), 2010. Piscataway: IEEE; 2010. p. 657–668.
18. Liu H, Orban D. Gridbatch: cloud computing for large-scale data-intensive batch applications. In: 8th IEEE international symposium on cluster computing and the grid (CCGRID'08), 2008. Piscataway: IEEE; 2008. p. 295–305.
19. Barkhordari M, Niamanesh M. Arvand: a method to integrate multidimensional data sources into big data analytic structures. J Inf Sci Eng. 2018;34(2):505–18.
20. Atzeni P, Bugiottib F, Cabibboa L, Torlonea R. Data modeling in the NoSQL world. Comput Stand Interfaces. 2016.
21. Martinho B, Santos MY. An architecture for data warehousing in big data environments. In: International conference on research and practical issues of enterprise information systems. Cham: Springer; 2016. p. 237–250.
22. Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Zhang N, Murthy R. Hive-a petabyte scale data warehouse using hadoop. In: IEEE 26th international conference on data engineering (ICDE), 2010. Piscataway: IEEE; 2010. p. 996–1005.
23. Santos MY, Costa C. Data warehousing in big data: from multidimensional to tabular data models. In: Proceedings of the ninth international C* conference on computer science and software engineering. New York: ACM; 2016. p. 51–60.
24. Scabora LC, Brito JJ, Ciferri RR, Ciferri CDDA. Physical data warehouse design on NoSQL databases OLAP query processing over HBase. In: International conference on enterprise information systems, XVIII. Institute for systems and technologies of information, control and communication-INSTICC. 2016.
25. Dehdouh K, Bentayeb F, Boussaid O, Kabachi N. Using the column oriented NoSQL model for implementing big data warehouses. In: Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA). The Steering Committee of the World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). 2015; p. 469.
26. Chen S. Cheetah: a high performance, custom data warehouse on top of MapReduce. Proc VLDB Endow. 2010;3(1–2):1459–68.
27. He Y, Lee R, Huai Y, Shao Z, Jain N, Zhang X, Xu Z. RCFile: a fast and space-efficient data placement structure in MapReduce-based warehouse systems. In IEEE 27th international conference on data engineering (ICDE), 2011. Piscataway: IEEE; 2011. p. 1199–1208.
28. Floratou A, Patel JM, Shekita EJ, Tata S. Column-oriented storage techniques for MapReduce. Proc VLDB Endow. 2011;4(7):419–29.
29. Nykiel T, Potamias M, Mishra C, Kollios G, Koudas N. MRShare. Proc VLDB Endow. 2010;3(1–2):494–505.
30. Elghandour I, Aboulnaga A. ReStore. Proc VLDB Endow. 2012;5(6):587–97.
31. Abouzeid A, Bajda-Pawlikowski K, Abadi D, Silberschatz A, Rasin A. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. Proc VLDB Endow. 2009;2(1):922–33.
32. Vernica R et al.: Adaptive MapReduce using situation aware mappers. In: Proceedings of the 15th international conference on extending database technology. ACM; 2012.
33. Barkhordari M, Niamanesh M. ScaDiPaSi: an effective scalable and distributable MapReduce-based method to find patient similarity on huge healthcare networks. Big Data Res. 2015;2(1):19–27.
34. Kaldewey T, Shekita EJ, Tata S. Clydesdale: structured data processing on MapReduce. In: Proceedings of the 15th international conference on extending database technology. New York: ACM; 2012. p. 15–25.
35. Xiong X, Wenny BN, Wu A, Barnes WL, Salomonson VV. Aqua MODIS thermal emissive band on-orbit calibration, characterization, and performance. IEEE Trans Geosci Remote Sens. 2009;47(3):803–14.
36. Lee R, Luo T, Huai Y, Wang F, He Y, Zhang X. Ysmart: yet another sql-to-mapreduce translator. In: 31st international conference on distributed computing systems (ICDCS), 2011. Piscataway: IEEE; 2011. p. 25–36.
37. Agarwal S, Kandula S, Bruno N, Wu MC, Stoica I, Zhou J. Re-optimizing data-parallel computing. In: Proceedings of the 9th USENIX conference on networked systems design and implementation. Berkeley: USENIX Association; 2012. p. 21.
38. Agarwal S, Mozafari B, Panda A, Milner A, Madden S, Stoica I. BlinkDB: queries with bounded errors and bounded response times on very large data. In: Proceedings of the 8th ACM european conference on computer systems. New York: ACM; 2013.

39. Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S, Tzoumas K. Apache flink: Stream and batch processing in a single engine. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering. 2015;36(4).
40. Barkhordari M, Niamanesh M. Aras: a method with uniform distributed dataset to solve data warehouse problems for big data. Int J Distrib Syst Technol (IJDST). 2017;8(2):47–60.
41. Barkhordari M, Niamanesh M. Atrak: a MapReduce-based data warehouse for big data. J Supercomput. 2017;73(10):4596–610.
42. Barkhordari M, Niamanesh M. Hengam: a MapReduce-based distributed data warehouse for big data. Inter J Artif Life Res. 2018;8(1):16–35.
43. Hall A, Bachmann O, Büssow R, Gănceanu S, Nunkesser M. Processing a trillion cells per mouse click. Proc VLDB Endow. 2012;5(11):1436–46.
44. Engle C, Lupher A, Xin R, Zaharia M, Franklin MJ, Shenker S, Stoica I. Shark: fast data analysis using coarse-grained distributed memory. In: Proceedings of the 2012 ACM SIGMOD international conference on management of data. New York: ACM; 2012. p. 689–692.
45. Shinnar A, Cunningham D, Saraswat V, Herta B. M3R. Proc VLDB Endow. 2012;5(12):1736–47.
46. Armbrust M, Xin RS, Lian C, Huai Y, Liu D, Bradley JK, Zaharia M. Spark sql: relational data processing in spark. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. New York: ACM; 2015. p. 1383–1394.